Buttons

Thomas Schwarz

Buttons

- Apps have buttons
  - You press on them, and something happens
- Implementation in TkInter:
  - Create button (usually with text, sometimes with an image)
    - Always linked with an event handler
  - Place button
  - Create event handler — a “callback” function

Buttons

- A super-simple example: Create an app

button0.py - /Users/thomasschwarz/Documents/My website/Classes/Pytho

```
import tkinter as tk

class My_App:
    def __init__(self):
        self.main = tk.Tk()
        self.main.title("Buttons")
        self.create_widgets()
        self.main.mainloop()

    def create_widgets(self):
        self.label = tk.Label(text = "Press a button")
        self.label.pack(side = "top")

my_app = My_App()
```

Buttons

- Now create two buttons:
- Need to give a function as the command parameter
- Easiest to define as class parameters

button0.py - /Users/thomasschwarz/Documents/My website/Classes/

```
import tkinter as tk

class My_App:
    def __init__(self):
        self.main = tk.Tk()
        self.main.title("Buttons")
        self.create_widgets()
        self.main.mainloop()

    def create_widgets(self):
        self.label = tk.Label(text = "Press a button")
        self.label.pack(side = "top")
        self.button1 = tk.Button(self.main, text="Button 1",
                                 command = My_App.callback1)
        self.button1.pack(side="left")
        self.button2 = tk.Button(self.main, text="Button 2",
                                 command = My_App.callback2)
        self.button2.pack(side="right")

    def callback1():
        print("Button 1 has been pressed")

    def callback2():
        print("Button 2 has been pressed")

my_app = My_App()
```

Buttons

- Callbacks: Our code tells the button Constructor what to do in the future, namely when the button is pressed
- Small problem: we pass a function without parameters

Buttons

- If we want the button to do something to the app, the function needs to know how to reach the components
  - Solution:
  - Create only class fields instead of instance fields
  - This way everything is reachable from within function definitions

```
class MyApp:
    def __init__(self):
        MyApp.main = tk.Tk()
        MyApp.main.title("Buttons")
        self.create_widgets()
        MyApp.main.mainloop()
```

Buttons

- Change the background color of the main window
  - Uses the configure method and sets the parameter background

```
def callback1():
    print("callback 1 called")
    MyApp.main.configure(background="Red")
```

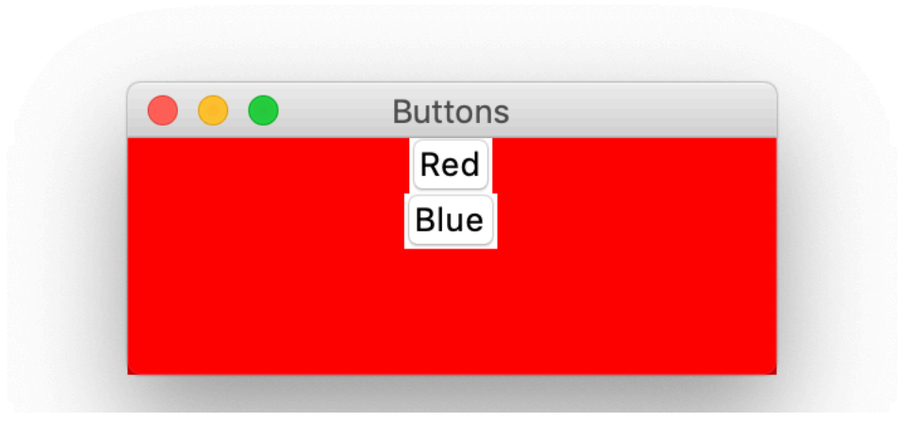
Buttons

button1.py - /Users/thomasschwarz/Documents/My website/Classes/Python_Big_...

```
import tkinter as tk

class MyApp:
    def __init__(self):
        MyApp.main = tk.Tk()
        MyApp.main.title("Buttons")
        self.create_widgets()
        MyApp.main.mainloop()
    def create_widgets(self):
        my_button1 = tk.Button(MyApp.main, text="Red", command=MyApp.callback1)
        my_button1.pack()
        my_button2 = tk.Button(MyApp.main, text="Blue", command=MyApp.callback2)
        my_button2.pack()
    def callback1():
        print("callback 1 called")
        MyApp.main.configure(background="Red")
    def callback2():
        print("callback 2 called")
        MyApp.main.configure(background="Blue")

my_app = MyApp()
```



Buttons

- We can also change the text of a label.
  - A polyglot “Hello World” Application
  - Main widget is a label with text
  - Use width and height to make it big enough
    - Number interpreted as text lines

```
def create_widgets(self):
     MyApp.label = tk.Label(MyApp.main,
                            text="Hello World",
                            height = 5,
                            width = 75)
     MyApp.label.pack(side="left")
```

Buttons

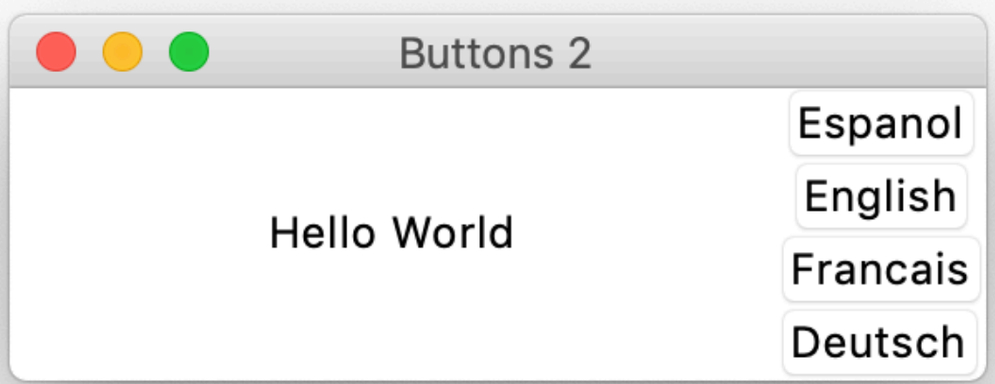
- Then create a number of buttons
  - Each would need their own callback function
    - But that is insane
    - Can use the lambda trick in order to call a function with different parameters
    - RECALL: lambda defines an anonymous python function
    - `lambda x, y: x+y` is the same as
    - ```
      def add(x, y):
          return x+y
      ```
      ```

button2.py - /Users/thomasschwarz/Documents/My website/Classes/Python_Big_Data/Module29/button2.

```
import tkinter as tk

class MyApp:
    def __init__(self):
        MyApp.main = tk.Tk()
        MyApp.main.title("Buttons 2")
        self.create_widgets()
        MyApp.main.mainloop()
    def create_widgets(self):
        MyApp.label = tk.Label(MyApp.main, text="Hello World", height=3, width=25)
        MyApp.label.pack(side="left")
        my_button1 = tk.Button(MyApp.main, text="Deutsch", command=MyApp.callback1)
        my_button1.pack(side="bottom")
        my_button2 = tk.Button(MyApp.main, text="Francais", command=MyApp.callback2)
        my_button2.pack(side="bottom")
        my_button3 = tk.Button(MyApp.main, text="English", command=MyApp.callback3)
        my_button3.pack(side="bottom")
        my_button4 = tk.Button(MyApp.main, text="Espanol", command=MyApp.callback4)
        my_button4.pack(side="bottom")
    def callback1():
        MyApp.label.configure(text="Hallo Welt")
    def callback2():
        MyApp.label.configure(text="Bonjour Monde")
    def callback3():
        MyApp.label.configure(text="Hello World")
    def callback4():
        MyApp.label.configure(text="Hola Mundo")

my_app = MyApp()
```



Buttons

- Define one callback function with an argument
- Define a function derived from that one anonymously

```
        my button4 = tk.Button(MyApp.main, text="Español",
                command=lambda : MyApp.callback("Hola Mundo"))
        my_button4.pack(side="bottom")
        my_button5 = tk.Button(MyApp.main, text="Italiano",
                command=lambda : MyApp.callback("Ciao Mondo"))
        my_button5.pack(side="bottom")
    def callback(my_text):
        MyApp.label.configure(text=my_text)
```

Buttons

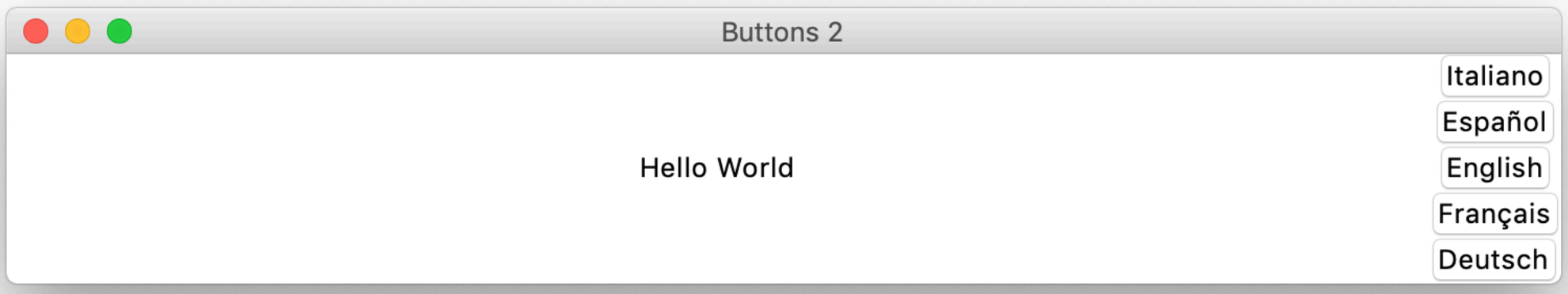
- Now we can go overboard and create many buttons

button3.py - /Users/thomasschwarz/Documents/My website/Classes/Python_Big_Data/Module29/button3.py (3.7.0)

```
import tkinter as tk

class MyApp:
    def __init__(self):
        MyApp.main = tk.Tk()
        MyApp.main.title("Buttons 2")
        self.create_widgets()
        MyApp.main.mainloop()
    def create_widgets(self):
        MyApp.label = tk.Label(MyApp.main, text="Hello World", height = 5, width = 75)
        MyApp.label.pack(side="left")
        my_button1 = tk.Button(MyApp.main, text="Deutsch", command=lambda : MyApp.callback("Hallo Welt"))
        my_button1.pack(side="bottom")
        my_button2 = tk.Button(MyApp.main, text="Français", command=lambda : MyApp.callback("Bonjour Le Monde"))
        my_button2.pack(side="bottom")
        my_button3 = tk.Button(MyApp.main, text="English", command=lambda : MyApp.callback("Hello World"))
        my_button3.pack(side="bottom")
        my_button4 = tk.Button(MyApp.main, text="Español", command=lambda : MyApp.callback("Hola Mundo"))
        my_button4.pack(side="bottom")
        my_button5 = tk.Button(MyApp.main, text="Italiano", command=lambda : MyApp.callback("Ciao Mondo"))
        my_button5.pack(side="bottom")
    def callback(my_text):
        MyApp.label.configure(text=my_text)

my_app = MyApp()
```



Ln: 26 Col: 0

The grid method

- Placing widgets with pack does not give a lot of control
  - Much more control wielded by grid
    - grid takes two coordinates, row and column
    - Distributes widgets into rows and columns
    - Can use rowspan or columnspan if a widget needs to take up more than a single row or column

grid method example

- We expand on the previous example in order to show how grid works
  - The label takes up several rows
    - But because it is big, it just defines a single big column
- The buttons are arranged in two columns
- By the way, Python 3 understands UTF, so we can add text in non-latin alphabets (russian, greek, gujarati, mahrati) as well as text with diacritic marks
  - I use copy and paste to convince the IDLE editor instead of looking up unicode codes.

```
import tkinter as tk

class MyApp:
    def __init__(self):
        MyApp.main = tk.Tk()
        MyApp.main.title("Buttons 3")
        self.create_widgets()
        MyApp.main.mainloop()
    def create_widgets(self):
        MyApp.label = tk.Label(MyApp.main, text="Hello World", height = 5, width = 25)
        MyApp.label.grid(rowspan=4, column = 0)
        my_button1 = tk.Button(MyApp.main, text="Deutsch", command=lambda : MyApp.callback("Hallo Welt"))
        my_button1.grid(row=0, column = 1)
        my_button2 = tk.Button(MyApp.main, text="Français", command=lambda : MyApp.callback("Bonjour Le Monde"))
        my_button2.grid(row=1, column = 1)
        my_button3 = tk.Button(MyApp.main, text="English", command=lambda : MyApp.callback("Hello World"))
        my_button3.grid(row=2, column = 1)
        my_button4 = tk.Button(MyApp.main, text="Español", command=lambda : MyApp.callback("Hola Mundo"))
        my_button4.grid(row=3, column = 1)
        my_button5 = tk.Button(MyApp.main, text="Italiano", command=lambda : MyApp.callback("Ciao Mondo"))
        my_button5.grid(row=4, column = 1)
        my_button6 = tk.Button(MyApp.main, text="русский", command=lambda : MyApp.callback("Привет, мир"))
        my_button6.grid(row=0, column = 2)
        my_button7 = tk.Button(MyApp.main, text="Ελληνική γλώσσα", command=lambda : MyApp.callback("Γειά σου Κόσμε"))
        my_button7.grid(row=1, column = 2)
        my_button8 = tk.Button(MyApp.main, text="ગુજરાતી", command=lambda : MyApp.callback("હેલો વર્લ્ડ"))
        my_button8.grid(row=2, column = 2)
        my_button9 = tk.Button(MyApp.main, text="मराठी", command=lambda : MyApp.callback("हॅलो वर्ल्ड"))
        my_button9.grid(row=3, column = 2)
        my_button10 = tk.Button(MyApp.main, text="Norsk", command=lambda : MyApp.callback("Hei Verden"))
        my_button10.grid(row=4, column = 2)
    def callback(my_text):
        MyApp.label.configure(text=my_text)

my_app = MyApp()
```